Computer Science

The Case For Prediction-based Best-effort Real-time Systems

Peter A. Dinda Bruce Lowekamp
Loukas Kallivokas David R. O'Hallaron

January 1999
CMU-CS-98-174

Carnegie Mellon

The Case For Prediction-based Best-effort Real-time Systems

Peter A. Dinda Bruce Lowekamp
Loukas Kallivokas David R. O'Hallaron

January 1999

CMU-CS-98-174

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

A version of this paper appeared in the Seventh Workshop on Parallel and Distributed Real-Time Systems (WPDRTS '99)

Abstract

We propose a prediction-based best-effort real-time service to support distributed, interactive applications in shared, unreserved computing environments. These applications have timing requirements, but can continue to function when deadlines are missed. In addition, they expose two kinds of adaptability: tasks can be run on any host, and their resource demands can be adjusted based on user-perceived quality. After defining this class of applications, we describe two significant examples, an earthquake visualization tool and a GIS map display tool, and show how they could benefit from the service. Finally, we present evidence that the service is feasible in the form of two studies of algorithms for host load prediction and for predictive task mapping.

Effort sponsored in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287, in part by the National Science Foundation under Grant CMS-9318163, and in part by a grant from the Intel Corporation.

Keywords: distributed real-time systems, interactive applications, host load prediction

1 Introduction

There is an interesting class of interactive applications that could benefit from a real-time service, but which must run on conventional reservation-less networks and hosts where traditional forms of real-time service are difficult or impossible to implement. However, these applications do not require either deterministic or statistical guarantees about the number of tasks that will meet their deadlines. Further, they are adaptable in two ways: their components are distributed, so a task can effectively be run on any available host, and they can adjust the computations tasks perform and the communication between tasks, trading off user-perceived degradation and the chances of meeting deadlines.

We propose a best-effort real-time service that uses history-based prediction to choose how to exploit these two levels of adaptation in order to cause most tasks to meet their deadlines and for the application to present reasonable quality to the user. We believe that such a service is feasible, and while providing no guarantees, would nonetheless simplify building responsive interactive applications and greatly improve user experience.

The paper has two main thrusts. The first is to define the class of resilient, adaptable, interactive applications we have described above and to show that it contains significant real applications. To this end, we analyze two applications in detail, explaining the demands users place on them and the adaptability they provide. The two applications are QuakeViz, a distributed visualization system for earthquake simulations being developed at CMU, and OpenMap, a framework for interactively presenting map-based information developed by BBN.

The second main thrust of the paper is to support the claim that history-based prediction is an effective way to implement a best-effort real-time service for this class of applications. We present two pieces of supporting evidence here. The first piece of supporting evidence is a trace-based simulation study of simple prediction algorithms that predict on which host a task is most likely to meet its deadline based on a history of past running times. One of the algorithms provides near optimal performance in the environments we simulated. The second piece of supporting evidence is a study of linear time series models for predicting host load that shows that simple, practical models can provide very good load predictions. Because running time is strongly correlated with the load experienced during execution, the fact that load is predictable suggests that information that can be collected in a scalable way can be used to make decisions about where to execute tasks. Together, these two results suggest that prediction is a feasible approach to implementing a best-effort real-time service.

2 Application characteristics

The applications we are interested in supporting have the four following key characteristics.

Interactivity

The applications are interactive—computation takes the form of tasks that are initiated or guided by a human being who desires responsiveness. Achieving responsiveness amounts to providing timely, consistent, and predictable feedback to individual user actions. If the feedback arrives too late or there is too much jitter for a series of similar actions, the utility of the program is degraded, perhaps severely. Research has shown that people have difficulty using an interactive application that does not respond in this manner [13, 19]. Our mechanism for specifying timely, consistent, predictable feedback is the task deadline.

Resilience

The applications are resilient in the face of missed deadlines to the degree that they do not require either statistical or deterministic guarantees from the real-time system. The inability to meet a deadline does not make these applications unusable, but merely results in lowered quality. For example, occasional missing frames in playing back video do not make the video performance unacceptable. Consistently missing or irregular frames, however, result in unacceptable playback. Resilience is the characteristic that suggests a best-effort real-time approach, instead of traditional "soft" (statistically guaranteed) [28, 7, 17] and "hard" (deterministically guaranteed) real-time approaches [25, 23].

Distributability

The applications have been developed with distributed, possibly parallel, operation in mind. We assume that it is possible to execute their tasks on any of the available hosts using, for example, mechanisms such as CORBA [21] or Java RMI [26]. Tasks need not be replicable (stateless), but any data movement required to execute a task on a particular host must be exposed, perhaps via an interface repository or reflection mechanism.

Adaptability

The applications expose controls, called *application-specific quality parameters*, that can be adjusted to change the amount of computation and communication resources a task requires. Adjustments such as changing resolution, image quality, frame rate, and response time may be needed in order to deal with situations where a task's deadline cannot be met by choosing the appropriate host to run it, or when longer term changes in the available resources result in many tasks missing their deadlines. These parameters can also be used as "knobs" to adjust the behavior of the application to meet the user's quality metrics.

3 QuakeViz

The Quake project developed a toolchain capable of detailed simulation of large geographic areas during strong earthquakes [2]. For example, a simulation was developed of the response of the San Fernando Valley to an aftershock of the 1994 Northridge Earthquake. The finite element representation of this simulation contains up to 77 million tetrahedrons producing over 40 million unknowns. Storing all of the data for each time step of the simulation would require approximately 6TB of data. Even selective output results in tens of gigabytes of data.

Thorough assessment of the seismicity associated with the geographic region requires accurate, interactive visualization of the data produced by the simulation. To facilitate rapid turn-around in improving the simulation and in testing different conditions, we are designing an environment

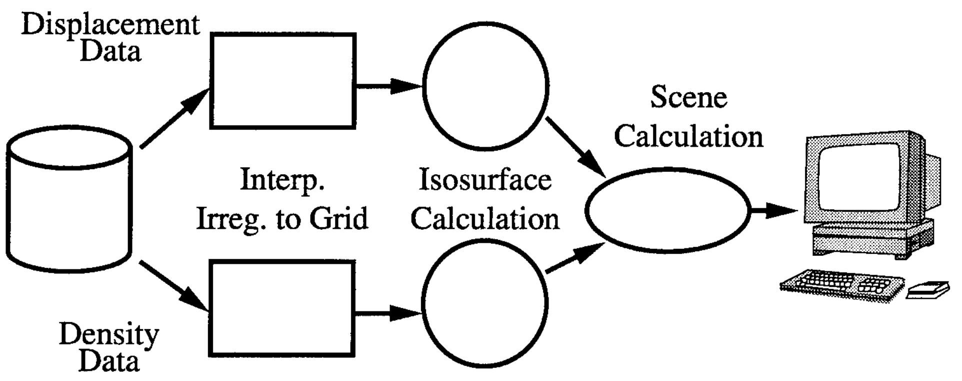


Figure 1: Stages of the Quake data visualization toolchain. Displacement data follows the upper path. Density data, which does not change during the simulation, but may need to be recalculated if the visualization parameters change, follows the lower path. Data can be downsampled on any edge if network bandwidth is limited.

where the visualization can be done on a researcher's regular desktop, without requiring special hardware support.

Designing a visualization system for Quake which can be used interactively on a standard desktop system across unreserved network resources is a challenge because these devices were not designed to handle this type of load. Nevertheless, with proper management of the visualization data, it is possible to achieve reasonable quality in this environment.

The visualization toolchain is shown in Figure 1. The raw simulation data is typically read from storage, rather than from the running application, because of the simulation's high cost, scheduling complexities, and the lack of any runtime tunable parameters in the simulation. The raw irregular mesh data is first interpolated onto a regular grid to facilitate processing and downsampled to a more manageable size. The resulting grid is then used to calculate isosurfaces corresponding to various response intensities. The displacement and density isosurfaces are combined with topology information to produce a 3-dimensional image, which is then drawn on the user's desktop display.

Because the Quake visualization is done offline, the user's input consists of controlling the parameters of the visualization. These operations may include rotating the display, removing surface layers, zooming in and out, and adjusting the parameters for isosurface calculation.

In the visualization diagram shown in Figure 1, generally we expect the resources to retrieve and do the initial interpolation to be available on only a few high-performance machines, such as those commonly used at remote supercomputing facilities. Because the output device is determined by the user's location, the isosurface calculation and scene calculation are the two components which can be freely distributed to optimize performance.

3.1 Task location

The location of these two phases is governed by the combination of network bandwidth, and computing and graphics resources available to the user. In fact, there are reasons to divide the pipeline at any of the three remaining edges, depending on predicted processor and network capacity. Three examples are shown in Figure 2.

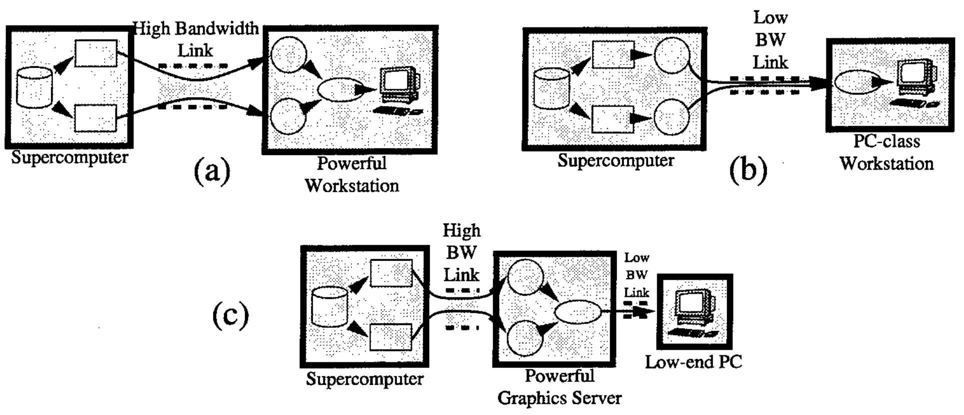


Figure 2: Different placements of the Quake tasks corresponding to different resource situations. (a) A high performance network and workstation are available for the visualization. (b) A low-bandwidth network and PC with a 3D graphics accelerator are used. (c) A low-end PC is used as a framebuffer displaying the result of a completely remote visualization process.

Figure 2a is the ideal case where resources allow the full regular mesh produced by the interpolation phase to be sent directly to the user's desktop. This would be possible in the case where there is a large amount of bandwidth between the remote site and the user's desktop and the user's desktop has sufficient power to perform the entire 3d processing operations in real-time. This situation is unlikely in all but the most powerful environments, but may offer the best opportunity for interactive use.

In Figure 2b, the isosurface calculation is performed in the supercomputer doing the interpolation. The scene calculation is done in the user's desktop machine. This could be used when less bandwidth is available, because the isosurfaces are significantly lower bandwidth than the grid data. The isosurface calculation is also fairly expensive, whereas the scene calculation can be done quite effectively by a variety of commodity graphics cards currently available.

A very limited case is shown in Figure 2c, where the user's desktop is acting only as a framebuffer for the images. This setup may be useful if the size of the final pixmap is smaller than the size of the 3D representation. This will depend on a number of factors, such as resolution, number of surfaces, translucency, and the occlusion of data by opaque geographic features. This arrangement could also be used in a case where the desktop machine does not support hardware 3D graphics.

3.2 Quality parameters

Frame rate, resolution, and interactive response time are the primary quality parameters which can be adjusted to meet the user's requirements. Location of the different tasks was discussed above. Now consider some of the other adaptations which can take place:

- While running uninterrupted, frame rate can be determined by the constant delays in the pipeline. If lower response latency is desired, the resolution of the images can be reduced at any edge of the pipeline. This may be important because there may be a high latency for issuing a change request all the way to the supercomputer processing the images.

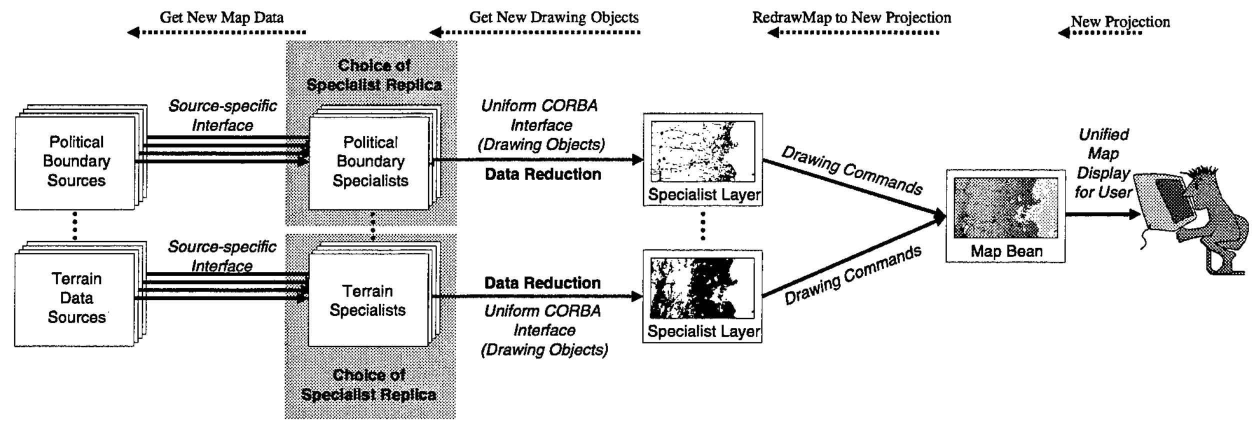

Get New Map Data
Get New Drawing Objects
RedrawMap to New Projection
New Projection
Choice of Specialist Replica
Political Boundary Sources
Source-specific Interface
Political Boundary Specialists
Uniform CORBA Interface (Drawing Objects)
Data Reduction
Specialist Layer
Drawing Commands
Map Bean
Unified Map Display for User
Terrain Data Sources
Source-specific Interface
Terrain Specialists
Data Reduction
Uniform CORBA Interface (Drawing Objects)
Specialist Layer
Drawing Commands
Choice of Specialist Replica

Figure 3: Structure of an OpenMap application. Solid arrows represent the flow of data, while dotted arrows represent user requests.

- Although zooming in on the data can be performed at any stage of the pipeline, it makes sense to propagate the change back to the interpolator, so that it can maximize the amount of data produced representing the region of interest. The combination of these adjustments makes it possible to provide quick response time to adjustments to the visualizer by temporarily reducing image quality until the adjustments can pass through the entire pipeline.

- Even if the system is running with the user's terminal being used only as a framebuffer, that image can be used for most adjustments in a temporary manner. For instance, although zooming in won't immediately reveal any more details, it will provide the user with the positive feedback required to maintain usability.

4 OpenMap

BBN's OpenMap is a architecture for combining geographical information from a variety of different, separately developed sources in order to present a unified coherent visual representation, in the form of a multi-layered map, to the end user [5, 18].

OpenMap consists of four different kinds of components. Geographical information is provided by third party *data sources*, which have unique interfaces. A *specialist* encapsulates a specific data source, hiding the details of accessing it behind a uniform CORBA interface. The interface is based on sequences of objects to be drawn. A specialist has a corresponding *layer* that draws an individual map based on the drawing objects. Finally, A *map bean* manages a group of layers, overlaying their maps to produce a single combined map for the user. Map beans and layers are Java Beans, which can be conveniently embedded into Java applications.

In Figure 3, we show the structure of an example OpenMap application where information from separate terrain and political boundary data sources are combined to present the user with a map of the Boston area. While the structure shown in Figure 3 appears at first glance to be a pipeline, it is important to note that it actually operates in a request-response manner. Computation happens only when the user decides to change the *projection* of the map (the set of layers and the region of the planet that is being viewed.)

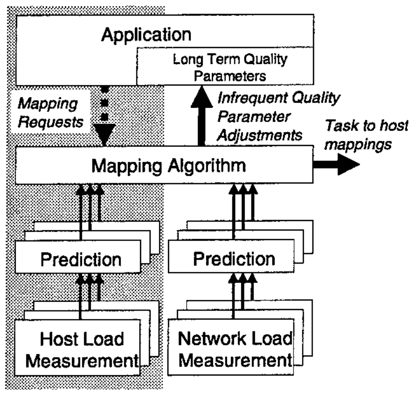


Figure 4: Structure of best-effort real-time system. The highlighted portion is discussed.

OpenMap is interactive—computation happens as a direct result of a projection change. To provide a good user experience, the time from a projection change to the resulting map display should be short, consistent, and predictable. A good abstraction for this requirement is a deadline placed on the computation initiated by a projection change. Achieving such deadlines is challenging because specialists and data sources may be located at distant sites and run on shared, unreserved hosts communicating via the Internet. However, missing OpenMap deadlines only degrades the user's experience—OpenMap is resilient.

The components of OpenMap were designed from the start to be physically distributed using CORBA communication mechanisms. We can use this enabler to build replicated specialists and data sources, as we highlight in gray in Figure 3. This provides a choice of which specialist is used to satisfy a projection change for a given layer, which is one of the degrees of freedom that a best-effort real-time system can exploit.

An OpenMap specialist can also expose two quality parameters to a best-effort real-time system. First, it can adjust the query sent to its data source to change the amount of computation that needs to be done to satisfy the request and the size of the response. This is a useful adaptation for when all of the replicated data sources are simultaneously too busy. The second quality parameter a specialist can expose is the size of the response it sends to its layer. If the communication path from the specialist to the layer should become very constrained, the specialist can send a lower quality drawing.

5 Structure of a best-effort real-time system

Figure 4 illustrates the structure of our proposed best-effort real-time system. The boxes represent system components. Thin arrows represent the flow of measured and predicted information, thick arrows represent the control the system exerts over the application, and the dotted arrow represents the flow of application requests. We have highlighted the parts of the design that we discuss in this paper.

To be used with our system, an application must have the attributes we discussed in Section 2: the execution of the application is decomposable into tasks with well-defined inputs, outputs, and resource requirements, these tasks can be run on any of the available hosts, and the end-user's quality (and the computation and communication required) can be changed via application-specific quality parameters.

The system accepts requests to run tasks from the application and then uses a mapping algorithm to assign these tasks to the hosts where they are most likely to meet their deadlines. A task mapping request includes a description of the data the task needs, its resource requirements (either supplied by the application programmer or predicted based on past executions of the task), and the required deadline. The mapping algorithm uses predictions of the load on each of the available hosts and on the network to determine the expected running time of the task on each of the hosts, and then runs the task on one of the hosts (the target host) where it is likely to meet its deadline with high confidence. If no such host exists, the system can either map the task suboptimally, hoping that the resource crunch is transient behavior which should soon disappear, or adjust the quality parameters of the application to attempt to reduce the task's computation and/or communication requirements or to add more slack to the deadline.

The choice of target host and the choice of application-specific quality parameters are adaptation mechanisms that are used at different time scales. A target host is chosen for every task, but quality parameters are only changed when many tasks have failed to meet their deadlines, or when it becomes clear that sufficient resources exist to improve quality while still insuring that most deadlines can be met by task mapping. Intuitively, the system tries to meet deadlines by using adaptation mechanisms that do not affect the user's experience, falling back on mechanisms that do only on failure. Of course, the system can also make these adjustments pre-emptively based on predictions.

The performance of the system largely reflects the quality of its measurements and predictions. The measurement and prediction stages run continuously, taking periodic measurements and fitting predictive models to series of these measurements. For example, each host runs a daemon that measures the load with some periodicity. As the daemon produces measurements, they are passed to a prediction stage which uses them to improve its predictive model. When servicing a mapping request, the mapping algorithm requests predictions from these models. The predictions are in the form of a series of estimated future values of the sequence of measurements, annotated with estimates of their quality and measures of the past performance of the predictive model. Given the predictions, the mapping algorithm computes the running time of the task as a confidence interval whose length is determined by the quality measures of the prediction. Higher quality predictions lead to shorter confidence intervals, which makes it easier for the mapping algorithm to choose between its various options.

6 Evidence for a history-based prediction approach

Is history-based prediction a feasible approach to implementing a best-effort real-time service for the application domain we described earlier? In this section, we present two pieces of supporting evidence here that suggest that it is. The first is a trace-based simulation study of simple mapping algorithms that use past task running times to predict on which host a task is most likely to meet its deadline. The study shows that being able to map a task to any host in the system exposes significant opportunities to meet its deadline. Further, one of the algorithms provides near optimal performance in the environments we simulated. The second piece of evidence is a study of linear time series models for predicting host load. We found that simple, practical models can provide very good load predictions, and these good predictions lead to short confidence intervals on the expected running times of tasks, making it easier for a mapping algorithm to choose between the hosts. Network prediction is of considerably current interest, so we conclude with a short discussion of representative results from the literature.

6.1 Relationship of load and running time

The results in this section depend on the strong relationship that exists between host load and running time for CPU-bound tasks. We measure the host load as the Unix one-minute load average and sample it every second. We collected week-long traces of such measurements on 38 different machines in August 1997 and March 1998. We use these extensive traces to compute realistic running times for the simulation experiments of Section 6.2 and in Section 6.3 we directly predict them with an eye to using the predictions to estimate running times. A detailed statistical analysis of the traces is available in [11]. Considering load as a continuous signal $z(t)$ the relationship between load and running time $t_{running}$ is

$$\int_0^{t_{running}} \frac{1}{1+z(t)} dt = t_{nominal}$$

where $t_{nominal}$ is the running time of the task on a completely unloaded host. Notice that the integral simply computes the inverse of the average load during execution. Further details on how this relationship was derived and verified can be found in [12].

6.2 Simple prediction-based mapping algorithms

Intelligently exploiting the freedom of which host to map a task to can significantly increase the number of tasks that meet their deadlines compared to strategies such as always mapping to the same host or mapping to random hosts. We reached this conclusion by developing and evaluating a variety of simple mapping algorithms for mapping compute-bound tasks.

We developed and evaluated nine different mapping algorithms that use a past history of task running times on the different hosts to predict the host on which the current task's deadline is most likely to be met. These algorithms included several different window-average schemes, schemes based on confidence intervals computed using stationary IID stochastic models of running times, and simple neural networks. For conciseness, we will only discuss the most successful of the algorithms here. That algorithm, which we call *RangeCounter(W)*, associates a quality value, initially zero, with each host. To choose a host, the following steps are taken: If no host has a quality value significantly higher than zero, a random host is chosen, otherwise the host with the highest quality value is chosen. Next, the quality values of all hosts are reduced (aged) by a small amount. If the mapping results in the deadline being met, the chosen host's quality value is increased by the inverse of our statistical confidence in it. The statistical confidence is the cumulative probability over the deadline range of a Normal distribution parameterized by the sample mean and variance of previous task running times on the host. If the mapping is unsuccessful, we divide the chosen host's quality value by two. *RangeCounter(W)* rewards good choices based on how unlikely they

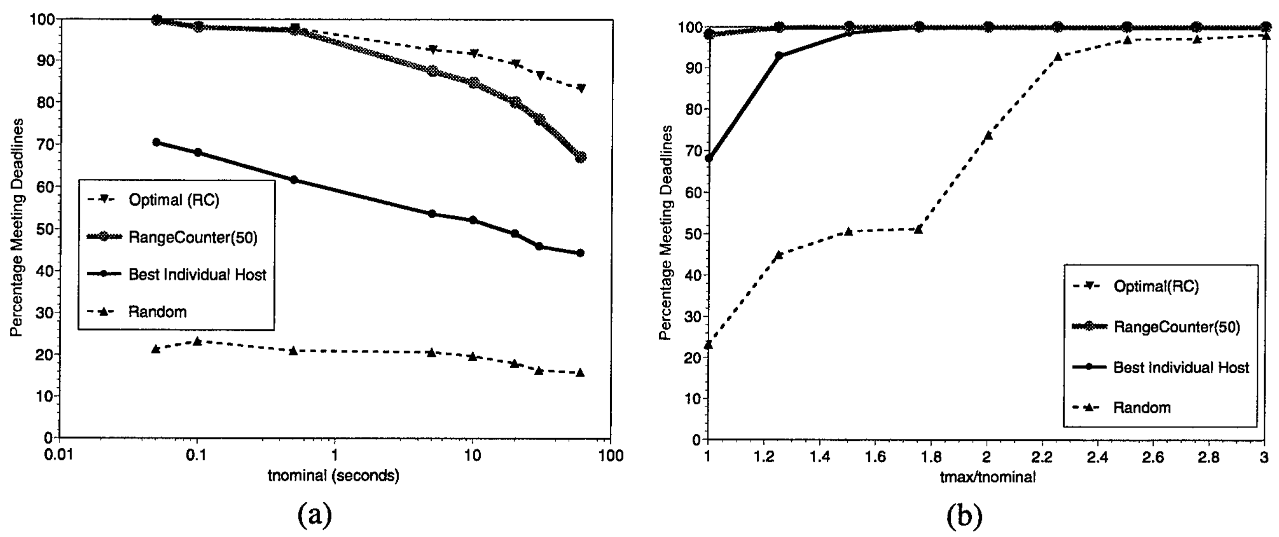


Figure 5: Representative data from our study of simple prediction-based mapping algorithms: (a) the effect of varying nominal execution time, (b) the effect of increasing deadline slack.

appear to be able to bear fruit and punishes bad choices equally. The idea is to encourage exploration and risk taking, but to react quickly and decisively to failure. The aging process discourages complacency.

We evaluated the algorithms with a trace-driven simulator that uses the load traces described in Section 6.1 to synthesize highly realistic execution times. The following loop was simulated:

```
for i=1 to N do
   MAP task() IN t_max
endfor
```

where the nominal execution time, $t_{nominal}$, of `task` is either constant or chosen from an enumerated distribution, and the deadline, t_{max} can be varied. Communication costs are ignored here—`task` is entirely compute-bound and requires no inputs or outputs to be communicated.

Given a test configuration of a group of hosts (represented by their load traces), the simulator determines how many tasks meet their deadlines for each algorithm. In addition, the simulator also evaluates a random mapping and determines the maximum (optimal) number of tasks that could have met their deadlines using all of the hosts, and using only the best individual host.

We studied a six different configurations of hosts, and for each one we used eight different values of the nominal execution time $t_{nominal}$, and twelve different deadlines t_{max}, ranging $t_{nominal}$ to three times $t_{nominal}$. We simulated $N = 100,000$ tasks for each of these 576 different combinations for a total of 57.6 million simulated tasks. An additional 72 combinations (7.2 million calls) representing varying $t_{nominal}$ were also simulated.

Figure 5(a), which is representative of our overall results, illustrates the effect of varying the nominal time, $t_{nominal}$ with a tight deadline of $t_{max} = t_{nominal}$. The configuration being studied contains 8 hosts with a wide variety of different load behaviors. The x-axis in the figure is the nominal time, $t_{nominal}$, and ranges from 50 milliseconds to 60 seconds, while the y-axis is the percentage of tasks that meet their deadlines. In addition to the performance of the *RangeCounter(W)* algorithm, the performance of random mapping, always mapping to the best individual host, and mapping optimally to all of the hosts are also plotted.

Notice that there is a substantial gap between the performance of the optimal mapping algorithm and the performance of random mappings. Further, it is clear that always mapping to the same host, even the best one does not result in an impressive number of tasks meeting their deadlines. These differences suggest that a good mapping algorithm can make a large difference. We can also see that *RangeCounter(W)* performs quite well, even for fairly long tasks.

Even with relaxed deadlines, a good mapping algorithm can make a significant difference. Figure 5(b), which is representative of our results, illustrates the effect of relaxing the deadline t_{max} (normalized to $t_{nominal}$ on the x-axis) on the percentage of tasks that meet their deadlines. As before, data is plotted for *RangeCounter(W)*, random mapping, always mapping to the best individual host, and optimal mapping. Notice that there is a large, persistent difference between random mapping and optimal mapping even with a great deal of slack. Further, using the best single host is suboptimal until the deadline is almost doubled. On the other hand, *RangeCounter(W)* presents nearly optimal performance even with the tightest possible deadline.

In a real system, the cost of communication will result in fewer deadlines being possible to be met and will tend to encourage the placement of tasks near the data they would consume, and thus possibly reduce the benefits of prediction. Still, this study shows that it is possible to meet many deadlines by using prediction to exploit the degree of freedom that being able to run a task on any host gives us.

6.3 Linear time series models for host load prediction

Implicitly predicting the current task's running time on a particular host based on previous task running times on that host, as we did in the previous section, limits scalability because each application must keep track of running times for every task on every node, and the predictions are all made on a single host.

Because running time is so strongly related to load, as we discussed in Section 6.1, another possibility is to have each host directly measure and predict its own load and then make these predictions available to other hosts. When mapping a task submitted on some host, the best-effort real-time system can use the load predictions and the resource requirements of the task to estimate its running time on each of the other hosts and then choose one where the task is likely to meet its deadline with high confidence.

Load predictions are unlikely to be perfect, so the estimates of running time are actually confidence intervals. Better load predictions lead to smaller confidence intervals, which makes it easier for the mapping algorithm to decide between the available hosts. We have found that very good load predictions that lead to acceptably small confidence intervals can be made using relatively simple linear time series models. Due to space limits, we do not discuss these models here, but interested readers will find Box, et al. [9] or Brockwell and Davis [10] to be worthy introductions. Figure 6 illustrates the benefits of such models on (a) a heavily loaded server and (b) a lightly loaded interactive cluster machine. In each of the graphs we plot the length of the confidence interval for the running time of a one second task as a function of how far ahead the load predictions are made. Figure 6(a) compares the confidence intervals for a predictive AR(9) model and for the raw variance of the load. Notice that for short-term predictions, the AR(9) model provides confi-

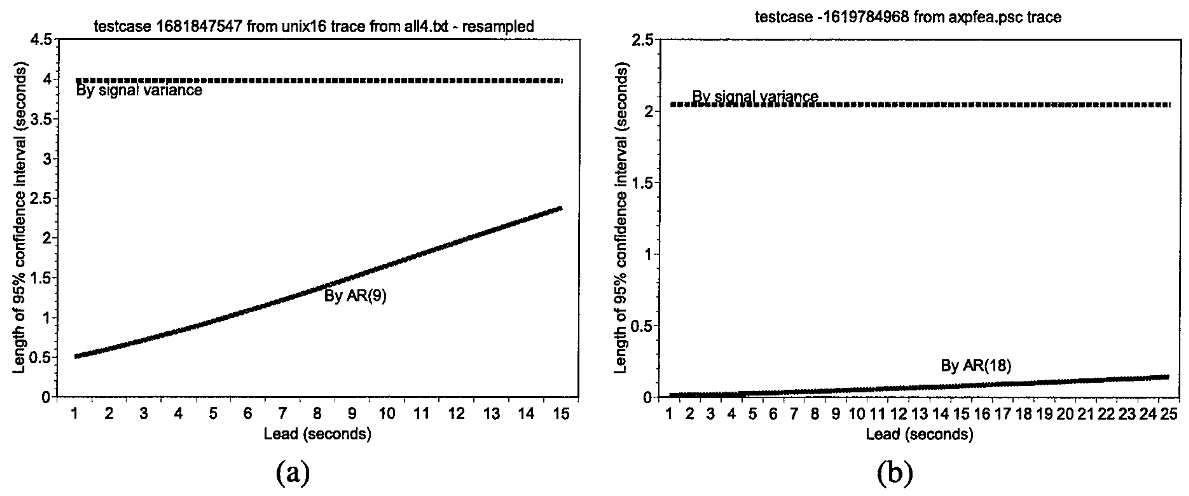


Figure 6: Benefits of prediction: (a) server machine with 40 users and long term load average of 10, (b) interactive cluster machine with long term load average of 0.17.

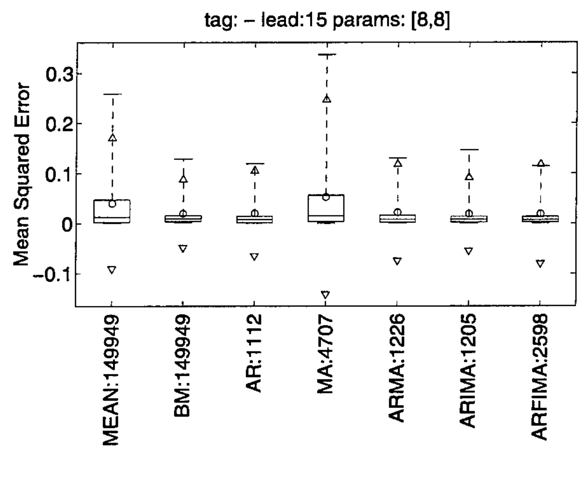
dence intervals that are almost an order of magnitude smaller than the raw signal. For example, when predicting 5 seconds ahead, the confidence interval for the AR(9) model is less than 1 second while the confidence interval for the raw load signal is about 4 seconds. Figure 6(b) shows that such benefits are also possible on a very lightly loaded machine.

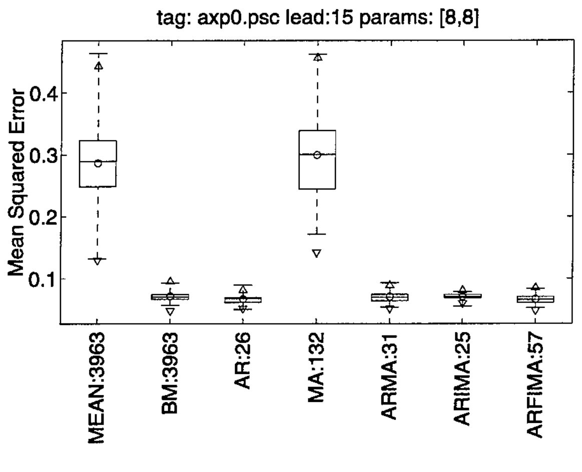
The prediction process begins by fitting a *model* to a history of periodically sampled load measurements. As new measurements become available, they are fed to the fitted model in order to refine its predictions. At any point after the model is fitted, we can request predictions for an arbitrary number of samples into the future. The quality of predictions for k samples into the future is measured by the *k-step ahead mean squared error*, which is the average of the squares of the differences between the k-step ahead predictions and their corresponding actual measurements. After some number of new samples have been incorporated, a decision is made to refit the model and the process starts again. We refer to one iteration of this process as a *testcase.*

A good model provides *consistent predictability* of load, by which we mean it satisfies the following two requirements. First, for the average testcase, the model must have a considerably lower expected mean squared error than the expected raw variance of the load. The second requirement is that this expectation is also very likely, or that there is little variability from testcase to testcase. Intuitively, the first requirement says that the model provides good predictions on average, while the second says that most predictions are close to that average.

In a previous paper [12], we evaluated the performance of linear time series models for predicting our load traces using the criterion of consistent predictability discussed above. Although load exhibits statistical properties such as self-similarity (Bassingwaighte, et al. [3] and Beran [6] provide good introductions to self-similarity and long-range dependence) and epochal behavior [11] that suggest that complex, expensive models such as ARFIMA [16, 14] or TAR [27] models might be necessary to ensure consistent predictability, we found that relatively simple models actually performed just about as well.

Figure 7(a) shows the performance of 8 parameter versions of the models we studied for 15 second ahead predictions, aggregated over all of our traces, while Figure 7(b) shows the performance on the trace of a single, moderately loaded interactive host. Each of the graphs in Figure 7 is a Box





(a) (b)

Figure 7: Performance of 8 parameter host load prediction models for 15 second ahead predictions: (a) All traces, (b) Moderately loaded interactive host.

plot that shows the distribution of 15-step-ahead (predicting load 15 seconds into the future) mean squared error. The data for the figure comes from running a large number of randomized testcases. Each testcase fits a model to a random section of a trace and then tests the model on a consecutive random section of the trace. In the figure, each category is a specific model and is annotated with the number of testcases used. For each model, the circle indicates the expected mean squared error, while the triangles indicated the 2.5th and 97.5th percentiles assuming a normal distribution. The center line of each box shows the median while the lower and upper limits of the box show the 25th and 75th percentiles and the lower and upper whiskers show the actual 2.5th and 97.5th percentiles.

Each of the predictive models has a significantly lower expected mean squared error than the expected raw variance of the load (measured by the MEAN model) and there is also far less variation in mean square error from testcase to testcase. These are the criteria for consistent predictability that we outlined earlier. Another important point is that there is little variation in the performance across the predictive models, other than that the MA model does not perform well. This is interesting because the ARMA, ARIMA, and especially the long-range dependence capturing ARFIMA models are vastly more expensive to fit and use than the simpler AR and BM models. An important conclusion of our study is that reasonably high order AR models are sufficient for predicting host load. We recommend AR(16) models or better.

6.4 Network prediction

Predicting network traffic levels is a challenging task due to the large numbers of machines which can create traffic over a shared network. Statistical models are providing a better understanding of how both wide-area [1, 22] and local-area [29] network traffic behave. Successes have been reported in using linear time series models to predict both long term [15] and short term [4] Internet traffic. These results and systems such as NWS [30] and Remos [20] are being developed to provide the predictions of network performance that are needed to provide best effort real-time service. For applications which are interested in predicting the behavior of a link on which they are already communicating on, passive monitoring may be appropriate [8, 24].

7 Conclusion

We have described two applications which can benefit from a best-effort real-time service. Without such a service, people using such interactive applications would face a choice between acquiring other, possibly reserved or dedicated resources or running the application at degraded quality. QuakeViz and OpenMap represent an interesting class of applications which people such as engineering researchers, and military and civilian GIS users can benefit from being able to use in their existing environments.

The success of best-effort real-time depends on the accuracy of the predictions of resource availability. We have shown that CPU load can be predicted with a high degree of accuracy using simple history-based time series models. When combined with currently available network information systems, these resources allow decisions to be made for locating tasks and selecting application parameters to provide a usable system.

We are currently integrating the CPU load prediction and network prediction into the Remos architecture. The development of this system and integration of QuakeViz and OpenMap to make use of such a system will allow us to test the success of best-effort real-time at meeting real-world quality demands in a variety of environments similar to how these applications are used by end-users.

Bibliography

[1] BALAKRISHNAN, H., STEMM, M., SESHAN, S., AND KATZ, R. H. Analyzing stability in wide-area network performance. In *Proceedings of SIGMETRICS'97* (1997), ACM, pp. 2–12.

[2] BAO, H., BIELAK, J., GHATTAS, O., KALLIVOKAS, L. F., O'HALLARON, D. R., SHEWCHUK, J. R., AND XU, J. Large-scale Simulation of Elastic Wave Propagation in Heterogeneous Media on Parallel Computers. *Computer Methods in Applied Mechanics and Engineering 152*, 1–2 (Jan. 1998), 85–102.

[3] BASSINGTHWAIGHTE, J. B., BEARD, D. A., PERCIVAL, D. B., AND RAYMOND, G. M. Fractal structures and processes. In *Chaos and the Changing Nature of Science and Medicine: An Introduction* (April 1995), D. E. Herbert, Ed., no. 376 in AIP Conference Proceedings, American Institute of Physics, pp. 54–79.

[4] BASU, S., MUKHERJEE, A., AND KLIVANSKY, S. Time series models for internet traffic. Tech. Rep. GIT-CC-95-27, College of Computing, Georgia Institute of Technology, February 1995.

[5] BBN CORPORATION. Distributed spatial technology laboratories: Openmap. (web page). http://javamap.bbn.com/.

[6] BERAN, J. Statistical methods for data with long-range dependence. *Statistical Science 7*, 4 (1992), 404–427.

[7] BESTAVROS, A., AND SPARTIOTIS, D. Probabilistic job scheduling for distributed real-time applications. In *Proceedings of the First IEEE Workshop on Real-Time Applications* (May 1993).

[8] BOLLIGER, J., GROSS, T., AND HENGARTNER, U. Bandwidth modelling for network-aware applications. In *Proceedings of Infocomm'99* (1999). to appear.

[9] BOX, G. E. P., JENKINS, G. M., AND REINSEL, G. *Time Series Analysis: Forecasting and Control*, 3rd ed. Prentice Hall, 1994.

[10] BROCKWELL, P. J., AND DAVIS, R. A. *Introduction to Time Series and Forecasting*. Springer-Verlag, 1996.

[11] DINDA, P. A. The statistical properties of host load. In *Proc. of 4th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR'98)* (Pittsburgh, PA, 1998), vol. 1511 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 319–334. Extended version available as CMU Technical Report CMU-CS-TR-98-143.

[12] DINDA, P. A., AND O'HALLARON, D. R. An evaluation of linear models for host load prediction. Tech. Rep. CMU-CS-TR-98-148, School of Computer Science, Carnegie Mellon University, November 1998.

[13] EMBLEY, D. W., AND NAGY, G. Behavioral aspects of text editors. *ACM Computing Surveys 13*, 1 (January 1981), 33–70.

[14] GRANGER, C. W. J., AND JOYEUX, R. An introduction to long-memory time series models and fractional differencing. *Journal of Time Series Analysis 1*, 1 (1980), 15–29.

[15] GROSCHWITZ, N. C., AND POLYZOS, G. C. A time series model of long-term NSFNET backbone traffic. In *Proceedings of the IEEE International Conference on Communications (ICC'94)* (May 1994), vol. 3, pp. 1400–4.

[16] HOSKING, J. R. M. Fractional differencing. *Biometrika 68*, 1 (1981), 165–176.

[17] JENSEN, E. D., LOCK, C. D., AND TOKUDA, H. A time-driven scheduling model for real-time operating systems. In *Proceedings of the Real-Time Systems Symposium* (Februrary 1985), pp. 112–122.

[18] KARR, D. A., BAKKEN, D. E., ZINKY, J. A., AND LAWRENCE, T. F. Towards quality of service for groupware. BBN Corporation (Submitted to ICDCS '99).

[19] KOMATSUBARA, A. Psychological upper and lower limits of system response time and user's preferance on skill level. In *Proceedings of the 7th International Conference on Human Computer Interaction (HCI International 97)* (August 1997), G. Salvendy, M. J. Smith, and R. J. Koubek, Eds., vol. 1, IEE, pp. 829–832.

[20] LOWEKAMP, B., MILLER, N., SUTHERLAND, D., GROSS, T., STEENKISTE, P., AND SUBHLOK, J. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 1998), IEEE, pp. 189–196.

[21] OBJECT MANAGEMENT GROUP. The common object request broker: Architecture and specification. Tech. Rep. Version 2.0, Object Management Group, July 1995.

[22] PAXSON, V., AND FLOYD, S. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking 3*, 3 (June 1995), 226–244.

[23] SCHWABL, W., REISINGER, J., AND GRUENSTEIDL, G. A survey of MARS. Tech. Rep. 16/89, Technische Universitaet Wien, October 1989.

[24] SESHAN, S., STEMM, M., AND KATZ, R. H. SPAND: Shared passing network performance discovery. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems* (December 1997), pp. 135–46.

[25] STANKOVIC, J., AND RAMAMRITHAM, K. *Hard Real-Time Systems*. IEEE Computer Society Press, 1988.

[26] SUN MICROSYSTEMS, INC. Java remote method invocation specification, 1997. Available via http://java.sun.com.

[27] TONG, H. *Threshold Models in Non-linear Time Series Analysis*. No. 21 in Lecture Notes in Statistics. Springer-Verlag, 1983.

[28] WALDSPURGER, C. A., AND WEIHL, W. E. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the First Symposium on Operating Systems Design and Implementation* (1994), Usenix.

[29] WILLINGER, W., MURAD S, T., SHERMAN, R., AND WILSON, D. V. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. In *Proceedings of ACM SIGCOMM '95* (1995), pp. 100–113.

[30] WOLSKI, R. Dynamically forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC)* (August 1997).

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890